Computation of winning strategies for μ-Calculus by fixpoint iteration

Christian Neukirchen

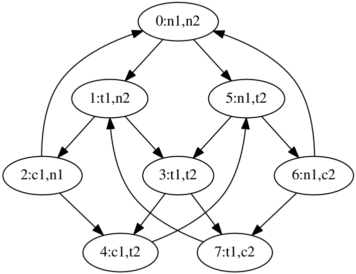
TCS Oberseminar · July 11, 2014

Overview

- Short introduction to μ-calculus
- Parity games and strategies
- Strategies for μ-calculus
- Example: mutex
- Implementation and optimization
- Future

Labelled Transition Systems

We consider LTS having a non-empty set of states S, total relations $\xrightarrow{a} \in S \times S$ (for actions $a \in \mathcal{A}$) and propositions $p \in \mathcal{P}$ which hold at a state or not.



μ-calculus: Syntax (from Hofmann and Rueß 2014)

$$
\begin{array}{rcll}
\phi & ::= & X & \\
 & | & p \ \ | \ \ \neg p & \\
 & | & [a]\phi & \textit{(for all } a\textit{-transitions)} \\
 & | & \langle a \rangle \phi & \textit{(} a\textit{-transition exists)} \\
 & | & \phi_1 \wedge \phi_2 \ \ | \ \ \phi_1 \vee \phi_2 & \\
 & | & \mu X.\phi & \textit{(least fixpoint)} \\
 & | & \nu X.\phi & \textit{(greatest fixpoint)}
\end{array}
$$

Only propositions can be negated to ensure monotonicity.

μ-calculus: Examples

"q holds everywhere along the path"

$$\nu X. q \wedge [a] X$$

"q holds infinitely often on the path"

$$\nu X. \mu Y. (q \wedge \langle a \rangle X) \vee \langle a \rangle Y$$

"p holds at every even position" (more powerful than CTL*)

$$\nu X. p \wedge \langle a \rangle \langle a \rangle X$$

μ-calculus: Set semantics

$$
\begin{aligned}
sem(X, \eta) &= \eta(X) \\
sem(\phi_1 \wedge \phi_2, \eta) &= sem(\phi_1, \eta) \cap sem(\phi_2, \eta) \\
sem(\phi_1 \vee \phi_2, \eta) &= sem(\phi_1, \eta) \cup sem(\phi_2, \eta) \\
sem([a]\phi, \eta) &= \widetilde{pre}(\xrightarrow{a})(sem(\phi, \eta)) \\
sem(\langle a \rangle \phi, \eta) &= pre(\xrightarrow{a})(sem(\phi, \eta)) \\
sem(\mu X.\phi, \eta) &= iter_X(\phi, \eta, \emptyset) \\
sem(\nu X.\phi, \eta) &= iter_X(\phi, \eta, S) \\
s \in \widetilde{pre}(\xrightarrow{a})(U) &\Leftrightarrow \forall t \in S.\, s \xrightarrow{a} t \implies t \in U \\
s \in pre(\xrightarrow{a})(U) &\Leftrightarrow \exists t \in S.\, s \xrightarrow{a} t \wedge t \in U \\
iter_X(\phi, \eta, U) &= \text{let } U' := sem(\phi, \eta[X := U]) \text{ in} \\
&\qquad \text{if } U = U' \text{ then } U \text{ else } iter_X(\phi, \eta, U')
\end{aligned}
$$

Parity games

A *parity game* consists of a disjoint sum of positions $\mathsf{Pos} = \mathsf{Pos}_0 \cup \mathsf{Pos}_1$, a total edge relation $\rightarrow \subseteq \mathsf{Pos} \times \mathsf{Pos}$ and a priority function $\Omega : \mathsf{Pos} \rightarrow \mathbb{N}$.

Moves happen along the edge relation. The destination decides who moves next.

The game is *won* if the largest priority that occurs infinitely often is even, the opponent wins if it is odd.

Strategies for parity games

A *strategy* ρ is a function that tells the player how to move next.

A *positional strategy* only takes the the current position into account.

A position is in a *winning set* W_i if there exists a strategy ρ such that player i wins, starting at a position in W_i.

Theorem 1. Every position p is either in W_0 or W_1 and player i wins positionally from every position in W_i.

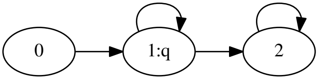
Strategies for μ-calculus

We can interpret a μ-calculus formula ϕ as a parity game. Moves can happen along the subformulae (example next slide). The priority of a position depends on the kind of formula and its nesting depth.

A partial winning strategy for μ-calculus is a partial function

$$
\begin{array}{rlr}
\Sigma : \Phi \times S \rightharpoonup & s & \textit{(move to state } s \in S\textit{)} \\
| & 1 & \textit{(take the left formula)} \\
| & 2 & \textit{(take the right formula)} \\
| & * & \textit{(take the only formula)}
\end{array}
$$

Small strategy example



$$\nu X.\mu Y.(q \wedge \langle a \rangle X) \vee \langle a \rangle Y$$

$$X := \nu X.Y$$

$$Y := \mu Y.(q \wedge \langle a \rangle X) \vee \langle a \rangle Y$$

```
X 0 -> * |
X 1 -> * |
Y 0 -> * |
Y 1 -> * |
q 1 -> * |
(<a>X) 0 -> X 1 |
(<a>X) 1 -> X 1 |
(<a>Y) 0 -> Y 1 |
(<a>Y) 1 -> Y 1 |
q /\ (<a>X) 1 -> * |
(q /\ (<a>X)) \/ (<a>Y) 0 -> #2 |
(q /\ (<a>X)) \/ (<a>Y) 1 -> #1
```

Updating strategies

Given two winning strategies Σ and Σ', we can define the partial winning strategy $\Sigma + \Sigma'$ as

$$\begin{aligned}(\Sigma + \Sigma')(\phi, s) = \text{if } &(\phi, s) \in dom(\Sigma)\\ \text{then } &\Sigma(\phi, s)\\ \text{else } &\Sigma'(\phi, s)\end{aligned}$$

Strategy semantics

$$
\begin{aligned}
\mathsf{SEM}(X)_\eta &= \{(X,s) \mapsto * \mid s \in \eta(X)\}\\
\mathsf{SEM}(p)_\eta &= \{(p,s) \mapsto * \mid p \text{ holds at } s\}\\
\mathsf{SEM}(\neg p)_\eta &= \{(p,s) \mapsto * \mid p \text{ does not hold at } s\}\\
\mathsf{SEM}(\phi \wedge \psi)_\eta &= \mathsf{SEM}(\phi)_\eta + \mathsf{SEM}(\psi)_\eta\\
&\quad + \{(\phi \wedge \psi, s) \mapsto * \mid (\phi, s) \in dom(\mathsf{SEM}(\phi)_\eta)\\
&\qquad\qquad\qquad\qquad\qquad \wedge (\psi, s) \in dom(\mathsf{SEM}(\psi)_\eta)\}\\
\mathsf{SEM}(\phi \vee \psi)_\eta &= \mathsf{SEM}(\phi)_\eta + \mathsf{SEM}(\psi)_\eta\\
&\quad + \{(\phi \vee \psi, s) \mapsto 1 \mid (\phi, s) \in dom(\mathsf{SEM}(\phi)_\eta)\}\\
&\quad + \{(\phi \vee \psi, s) \mapsto 2 \mid (\psi, s) \in dom(\mathsf{SEM}(\psi)_\eta)\}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{SEM}([a]\phi)_\eta &= \mathsf{SEM}(\phi)_\eta \\
&\quad + \{([a]\phi, s) \mapsto * \mid (\phi, s) \in dom(\mathsf{SEM}(\phi))_\eta\} \\
\mathsf{SEM}(\langle a\rangle\phi)_\eta &= \mathsf{SEM}(\phi)_\eta \\
&\quad + \{(\langle a\rangle\phi, s) \mapsto s' \mid s \xrightarrow{a} s' \wedge (\phi, s') \in dom(\mathsf{SEM}(\phi))_\eta\} \\
\mathsf{SEM}(\nu X.\phi)_\eta &= \mathsf{SEM}(\phi)_{\eta[X := sem(\phi,\eta)]} \\
\mathsf{SEM}(\mu X.\phi)_\eta &= \mathsf{ITER}_X(\phi, \eta, \{\})
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{ITER}_X(\phi, \eta, \Sigma) &= \text{let } \Sigma' := \mathsf{SEM}(\phi)_{\eta[X := dom(\Sigma)]} \text{ in} \\
&\qquad \text{if } \Sigma = \Sigma' \text{ then } \Sigma \text{ else } \mathsf{ITER}_X(\phi, \eta, \Sigma')
\end{aligned}
$$

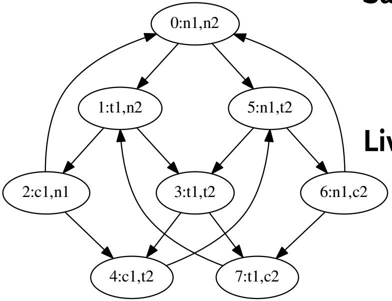
Checking strategies

Easy algorithm for checking whether a strategy is correct:

Run the strategy until you find a loop (hit the same formula at the same state again): then check whether the highest priority inside the loop is even (good) or odd (strategy is wrong!).

Can be implemented as simple recursive traversal.

Example: mutual exclusion (from Huth and Ryan 2004)



Safety "Only one process is in its critical section at any time."
$\nu Z.\neg(c_1 \wedge c_2) \wedge [a]Z$

Liveness "Whenever any process requests to enter its critical section, it will eventually be permitted to do so."
$\nu Z.(\neg t_1 \vee (\mu X.c_1 \vee [a]X)) \wedge [a]Z$

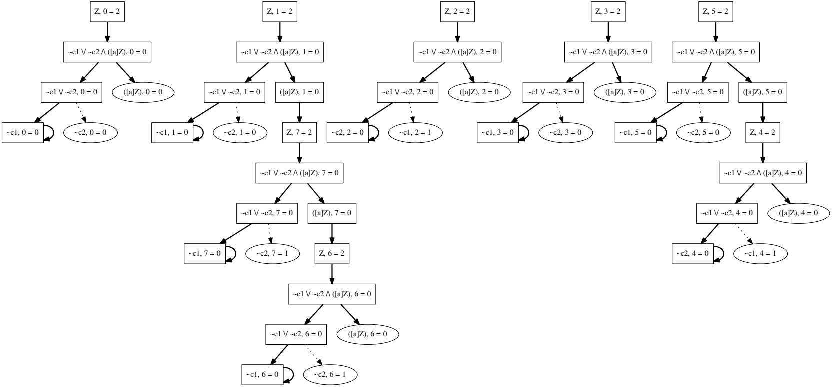
Mutex safety: sample run

```
% ./micromu.native huth-fig3.7.lts huth-fig3.7-safety.mu
Z =nu/1= ~c1 \/ ~c2 /\ ([a]Z)

### Execution time Mu.sem: 0.000000s
result:  0 1 2 3 4 5 6 7

### Execution time Strat.sem: 0.003333s
...

verifying for good state 0:
TRAV Z , 0 , loop-search
TRAV ~c1 \/ ~c2 /\ ([a]Z) , 0 , loop-search
TRAV ~c1 \/ ~c2 , 0 , loop-search
TRAV ~c1 , 0 , loop-search
```

```
TRAV ~c1 , 0 , loop-search
TRAV ~c1 , 0 , maxprio 0 here 0
TRAV ~c1 , 0 , maxprio 0 here 0
TRAV ([a]Z) , 0 , loop-search
state 0: true
...
verifying for good state 5:
TRAV Z , 5 , loop-search
TRAV ~c1 \/ ~c2 /\ ([a]Z) , 5 , loop-search
TRAV ~c1 \/ ~c2 , 5 , loop-search
TRAV ~c1 , 5 , loop-search
TRAV ~c1 , 5 , loop-search
TRAV ~c1 , 5 , maxprio 0 here 0
TRAV ~c1 , 5 , maxprio 0 here 0
TRAV ([a]Z) , 5 , loop-search
TRAV Z , 4 , loop-search
TRAV ~c1 \/ ~c2 /\ ([a]Z) , 4 , loop-search
```

```
TRAV ~c1 \/ ~c2 , 4 , loop-search
TRAV ~c2 , 4 , loop-search
TRAV ~c2 , 4 , loop-search
TRAV ~c2 , 4 , maxprio 0 here 0
TRAV ~c2 , 4 , maxprio 0 here 0
TRAV ([a]Z) , 4 , loop-search
state 5: true
verifying for good state 6:
state 6: known good
verifying for good state 7:
state 7: known good

### Execution time verify: 0.000000s
result:  0 1 2 3 4 5 6 7
verification passed
```

Mutex safety: strategy run



Mutex liveness: sample run

```
% ./micromu.native huth-fig3.7.lts huth-fig3.7-liveness.mu
X =mu/4= c1 \/ ([a]X)
Z =nu/1= ~t1 \/ X /\ ([a]Z)

### Execution time Mu.sem: 0.000000s
result:
```

The formula holds nowhere: the strategy is empty. The mutex does not guarantee liveness. Why?

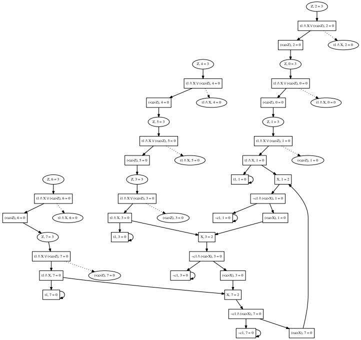
Mutex liveness: generating counterexamples

To generate a counter example, we can tell `micromu` to negate the formula using `-c`:

```
% ./micromu.native -c huth-fig3.7.lts huth-fig3.7-liveness.mu
### Execution time Mu.sem: 0.000000s
### Execution time Strat.sem: 0.003333s
### Execution time gendot: 0.006666s
### Execution time verify: 0.003333s
result:  0 1 2 3 4 5 6 7
verification passed
```

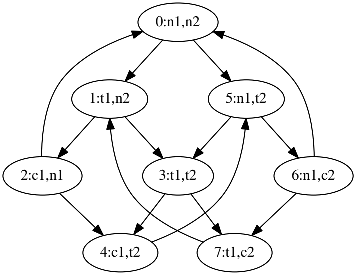
Looking at the strategy, we can find the counterexample.

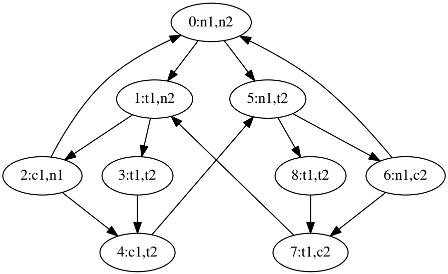
Mutex liveness: counterexample strategy



The big loop corresponds to $0 \to 1 \to 3 \to 7 \to 1 \to 3 \to 7 \to \cdots$

Fixing the mutex






```
% ./micromu.native huth-fig3.8.lts huth-fig3.7-liveness.mu
...
result:  0 1 2 3 4 5 6 7 8
verification passed
```

Implementation

compact: about 1 kLOC OCaml (another 1 kLOC thrown away during development), no external dependencies

quick: "worst-case" exponential example[1] $\mathcal{G}_1$ takes 0:02:40 and uses 28 MB RAM (down from 3+ hours / 6+ GB...)

simple, recursive algorithms: verifier should be easy to port to proof assistant (Coq)

---

[1] 15 nested quantifiers, cf. Friedmann 2009, section 5ff.

Systems of equations

First version actually substituted variables inside μ-formulae: consumes exponential amount of memory with nested formulae.

Rewrite equations as a ordered system of equations:

$$\nu Z.(\neg t_1 \vee (\mu X.c_1 \vee [a]X)) \wedge [a]Z$$

Need order to restore original formula.

$$Z \stackrel{\nu}{=} (\neg t_1 \vee X) \wedge [a]Z$$
$$X \stackrel{\mu}{=} c_1 \vee [a]X$$

Vastly simplifies implementation: makes ν-case trivial, μ-case a lot easier.

Optimizations

- Caching of results for $\nu X.\phi$ case
- Avoiding OCaml polymorphic compare (`compare_val`)
- Using maps for strategies, not association lists (requires careful strategy update)
- Caching of verified states (else easily quadratic runtime)
- Very helpful: *ocamlcp(1)*/*ocamlprof(1)* and *perf(1)*

Future: formal verification

- The checker is meant to be a *certified decision procedure*
- Formally verifying the checker to be correct results in a verified implementation of μ-calculus
- Done so far: definitions of least and greatest fixpoints (on arbitrary sets), specialized version of Knaster-Tarski, μ-calculus set semantics
- To do: serialize strategies into Coq terms
- To do: implement checker for strategies (using finite sets)
- To do: prove checker correct
- To do: extract verified checker?

Questions?

Thank you.

References

[1] Oliver Friedmann. "An Exponential Lower Bound for the Parity Game Strategy Improvement Algorithm as We Know it". In: *LICS*. IEEE Computer Society, 2009, pp. 145–156. ISBN: 978-0-7695-3746-7.

[2] Martin Hofmann and Harald Rueß. "Certification for mu-calculus with winning strategies". In: *ArXiv e-prints* (Jan. 2014). arXiv: 1401.1693 [cs.LO].

[3] Michael Huth and Mark Dermot Ryan. *Logic in Computer Science: Modelling and Reasoning About Systems*. 2nd. New York, NY, USA: Cambridge University Press, 2004, pp. I–XIV, 1–427. ISBN: 052154310X.